Processing 12: Slime soccer

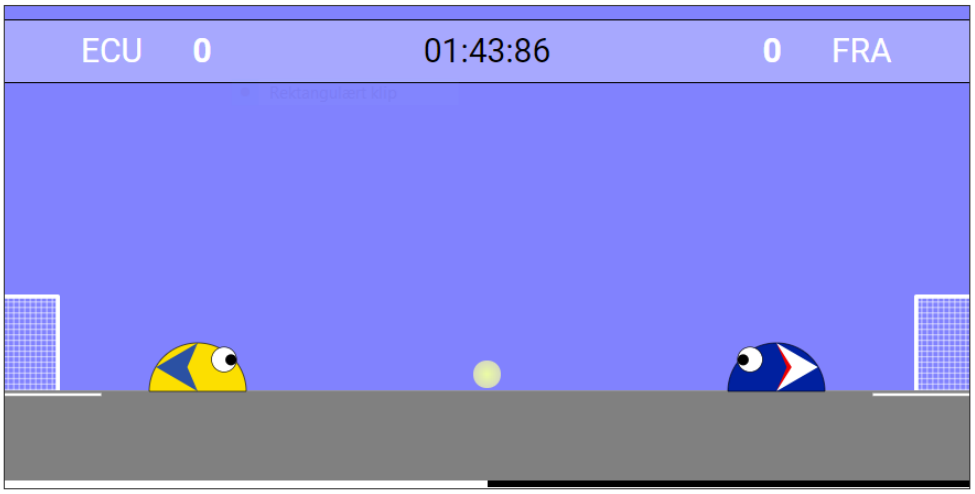
DDU, EUC Nord - HTX Hjørring
Version: andl, 10/2019

1 Slime soccer

Slime soccer er den bedste måde at findeud af, hvem der er bedst. Det bedste 2-player webbaserede flashspil blev uhyre populært i de senere 90'ere og op igennem nullerne. Dette projekt handler om Slime soccer.

Start med at gøre jer bekendte med spillet på `http://www.slime.cc/`. Spil evt. 2-player på friendly-mode.

Herefter kan I *fork* koden til en ufærdig implementation af spillet i Processing på Github `https://github.com/andlars/slime_soccer`. Fork endelig alle filerne og lav om I jeres readme, så det specificerer at det er et projektforløb, I deltager i. Tilføj alle gruppemedlemmer som collaborators på jeres repository på Github.com.



Figur 1: Det tætteste vi kommer på det originale Slime Soccer findes nok på `http://www.slime.cc/`. Dette brugte jeg (Anders) meget tid på i min gymnasietid...

class ball
x : PVector v : PVector r : float
render() update() bounce()

class slime
x : PVector v : PVector jump : boolean moveLeft : boolean moveRight : boolean
render() update()

Tabel 1: Klassediagrammer over `ball`- og `slime`-klasserne. Øverst er klassens navn, næste rum er reserveret til variable og til sidst angives klassernes metoder. For begge klasser er `render()`-metoden den metode, der udelukkende tegner objektet på canvasset ud fra sine variable, `update()` opdaterer objekternes positioner mens `ball`-klassens `bounce()`-metode får bolden til at hoppe på `slime`-objektet, givet dette input.

2 slime_soccer.pde

Processing sketchen består af 3 filer. Hovedfilen hedder `slime_soccer.pde`, mens der er to tillægsfiler, `ball.pde` og `slime.pde`. De to sidstnævnte filer indeholder `class`-specifikationer på klasserne af samme navne.

```
1  float g = 0.3;
2  ball b;
3  slime s;
4
5  void setup() {
6    size(1000, 600);
7    b = new ball();
8    s = new slime();
9  }
10
11 void draw() {
12   background(255);
13
14   s.render();
15   b.render();
16
17   s.update();
18   b.update();
19 }
20
21 void keyPressed() {
22   if (key == 'w') {
23     s.jump = true;
24   }
```

```
25   if (key == 'a') {
26     s.moveLeft = true;
27   }
28   if (key == 'd') {
29     s.moveRight = true;
30   }
31 }
32
33 void keyReleased() {
34   if (key == 'a') {
35     s.moveLeft = false;
36   }
37   if (key == 'd') {
38     s.moveRight = false;
39   }
40 }
```

Her er koden fra `ball.pde`

```
1  class ball {
2    PVector x, v;
3    float r;
4
5    ball() {
6      x = new PVector(width/2,height/2);
7      v = new PVector(0,0);
8      r = 40;
9    }
10
11   void render() {
12     noStroke();
13     fill(255, 0, 0);
14     ellipse(x.x, x.y, 2*r, 2*r);
15   }
16
17   void update() {
18     if (dist(x.x, x.y, s.x.x, s.x.y) < r + s.r) {
19       bounce(s);
20     } else {
21       v.mult(0.99);
22       v.y += g;
23     }
24
25     if (x.y == 0) {
26       v.mult(0.7);
27     }
28
29     v.limit(20);
30     x.add(v);
31
32     if (x.y + r > height) {
33       x.y = height - r;
```

```
    v.y = -v.y;
   }
  }

  void bounce(slime s) {
   PVector n = PVector.sub(x, s.x);
   float distanceCor = r + s.r - n.mag();
   n.normalize();
   v.sub(PVector.mult(n, 2*PVector.dot(n, v)));
   x.add(n.setMag(distanceCor));
   x.add(s.v);
   v.mult(1.5);
  }
}
```

Og her er koden fra `slime.pde`

```
class slime {
 PVector x, v;
 float r;
 boolean jump, moveLeft, moveRight;

 slime() {
  r = 100;
  x = new PVector(100, 100);
  v = new PVector(0, 0);
  jump = false;
  moveLeft = false;
  moveRight = false;
 }

 void render() {
  noStroke();
  fill(0, 255, 0);
  ellipse(x.x, x.y, 2*r, 2*r);
  fill(255);
  rect(x.x-r, x.y, 2*r, r);
 }

 void update() {
  v.y += g;

  if (moveLeft) {
   v.x += -8;
  }
  if (moveRight) {
   v.x += 8;
  }
  if (jump && x.y == height) {
   v.y = -10;
   jump = false;
  }
```

```
36
37      x.add(v);
38
39      if ( x.y > height) {
40        x.y = height;
41        v.y = 0;
42      }
43
44      v.x = 0;
45    }
46  }
```

Det kan være en god ide at gennemgå koden med hinanden inden I går i gang for at kunne redegøre for dens virkemåde.

3 Projektforløbet

I dette projekt skal I arbejde ved brug af samarbejdsmetoden scrum. Projektforløbet er stilladseret sådan at jeg har inddelt den tilgængelige tid i XX sprinter. Hver sprint er inddelt i en række delopgaver. Hver sprint indeholder følgende punkter:

1. Initieringsmøde hvor I...
   a) Sørger for at eventuelle branches på Github er samlet med jeres master. Dette er ønskværdigt, men ikke et must, da der kan være hængepartier fra tidligere sprints, der skal tages hånd om senere i projektforløbet.
   b) Alle delopgaverne er fordelt imellem gruppemedlemmerne.
   c) Beslutningerne er noteret i jeres logbog. Før jeres logbog som et tekstdokument på Github, så i alle kan skiftes til at være referent til disse møder.
2. Individuelt arbejde på jeres tildelte delopgaver. Opret hver delopgave som en branch fra morgenmødets master. I må selvfølgelig gerne supplere hinandens arbejde og/eller bytte delopgaver undervejs afhængigt af hvordan arbejdet skrider frem. Hver gang en delopgave bliver løst, opretter I en pull-request til master og merger, når det er muligt.
3. Et evalueringsmøde, hvor I opgør status på målet for sprintet, ajourfører logbogen samt evt. tilfører delopgaver til næste sprint.

Hver delopgave er farvekodet efter sværhedsgrad, så I kan benytte jer af hinandens styrker og svagheder, når I fordeler opgaverne imellem jer. Farvekoden er som følger:

- 🟢 Nemmest
- 🟡 Nem
- 🔴 Udfordrende, men ikke umulig

God fornøjelse!

3.1 Sprint 1

- 🟢 Variér værdien af den globale tyngdekraftsvariabel `g`, så dette bedre afspejler tyngdekraften i det faktiske spil.
- 🟢 Variér størrelsen på canvas, så dette bedre afspejler det faktiske spil.
- 🟢 Variér størrelsen på `slime`-objektet, så dette bedre afspejler det faktiske spil.
- 🟢 Variér størrelsen på `ball`-objektet, så dette bedre afspejler det faktiske spil.
- 🟡 Sørg for at `ball`-objektet hopper fra kanterne af canvas.

3.2 Sprint 2

- 🟢 Tegn øjne på `slime`-objektet.
- 🟡 Tegn et fodboldmål i venstre side af canvasset.
- 🟡 Opret en global variabel, `score`, der skal være af typen `int`. Der skal være en centreret tekst i toppen af canvasset, hvor `score` bliver vist sammen med lidt statisk tekst. Tjek evt. `https://processing.org/reference/text_.html`
- 🟢 Tegn omridset af `slime`-objekterne ved funktionskald til `arc()` i stedet for at tegne en cirkel med et rektangel ovenover. Læs om `arc()` på `https://processing.org/reference/arc_.html`

3.3 Sprint 3

- Sørg for at `score`-variablen tæller opad, når bolden befinder sig inden for målfeltet. Sæt evt. en *cooldown* på så `score` ikke tæller mange gange op, når der scores...
- Sørg for at `slime`-klassens øjne følger bolden. Få pupilen af øjet til at dreje sig efter boldens position.
- Sørg for at `slime`-klassens objekter ikke kan bevæge sig uden for canvasset.
- Tilføj en countdown i toppen af canvasset fra 2 minutter. Når de to minutter er gået skal spillet stoppes og den endelige score udskrives på skærmen.

3.4 Sprint 4

- Tilføj endnu en `slime` til spillet, så der er 2 spillere. Giv den nye `slime` piletasterne som controls. Tjek evt. `https://processing.org/reference/keyCode.html` for hjælp til dette. Husk at denne `slime` skal have øjnene i den anden side...
- Tilføj et mål i den anden side af canvasset. Hertil skal der også tilføjes en `score`-variabel, så den anden spiller kan få point. Sørg
- Nulstil spillet (alle variable og objekter, pånær `score`), når der scores, altså når `ball` befinder sig inden for målets position.
- Giv en `slime` et smil på, hvis de kommer mere end 3 mål foran.
- Gør sådan at der i starten af spillet tælles ned fra 3 sekunder inden spillet sættes i gang. Dette skal selvfølgelig også implementeres ved scoringer.
- Gør sådan at en `slime` kan holde bolden ved at holde deres respektive *ned*-taster nede (hhv. `s` og `keyCode DOWN`).

3.5 Sprint 5

Tilføj selv kosmetiske forbedringer til jeres spil eller tweake gameplay, fx hvis I har uløste bugs, som I ikke har løst undervejs.